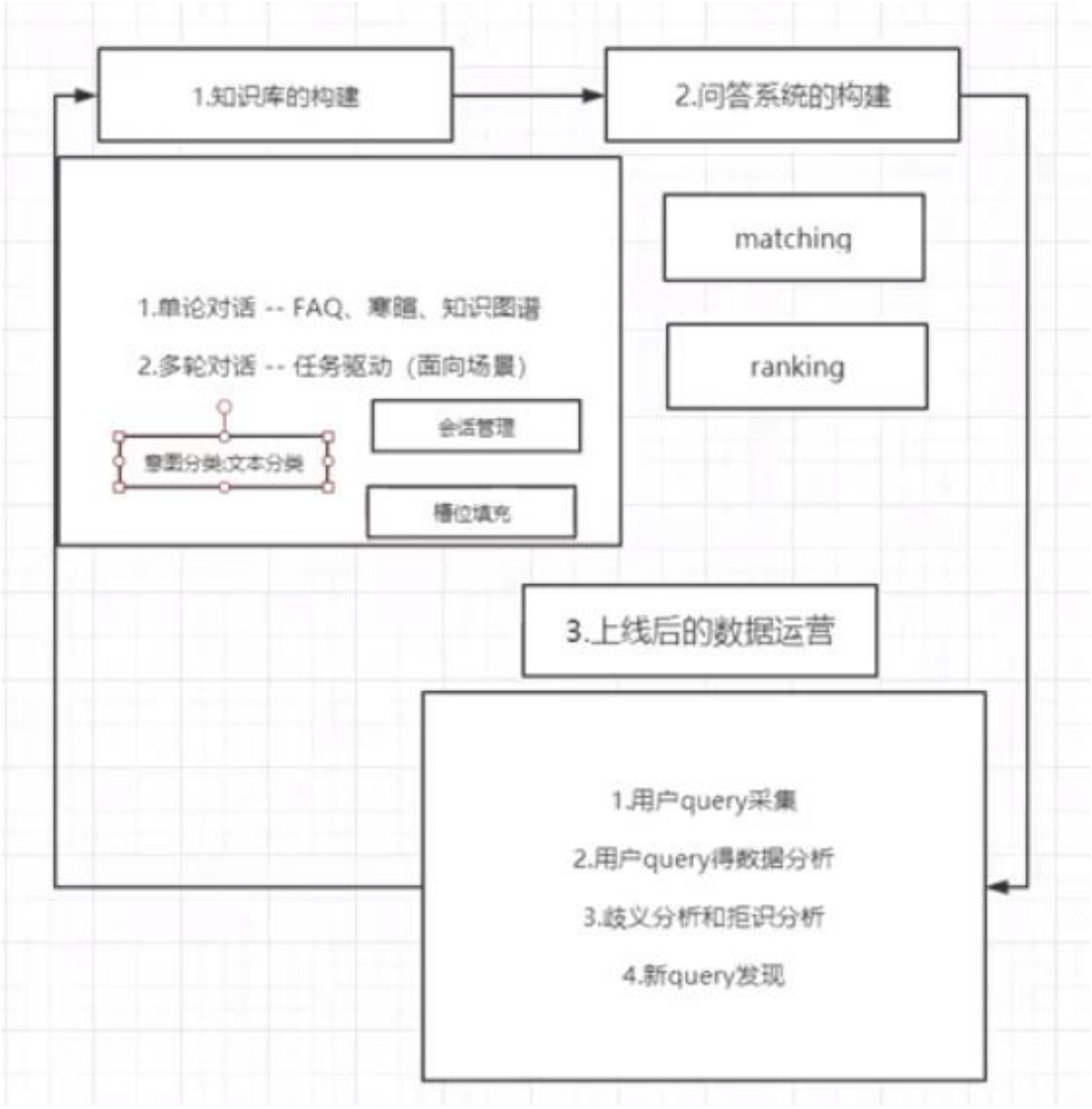
Aug. 22, 2019

1. 做作业（how to ge t tfidf）
2. 亲手推导 softmax, soidmoid，交叉熵等公示，然后自己写代码实现，最后用 sklearn 实现进行对比
3. 重新回顾准确率，召回率，F1
4. 对于 sigmoid 和 softmax 函数，其导数都为 sigmoid*(1-sigmoid)
5. 

6. 特征缩放有两种方法：1 是归一化（normalization） (x_i -min(x))/max(x) 2 是标准化(Standardization) (x_i-mean(x))/delta(x)

连续 regression,离散 regression（二分类，多分类 with 互斥和非互斥），word2vec，negative sampling，hierarchy softmax

7. 原文链接 https://blog.csdn.net/qq_23304241/article/details/80292859

卷积、池化和激活。卷积层是 CNN 网络的核心，激活函数帮助网络获得非线性特征，而池化的作用则体现在降采样：保留显著特征、降低特征维度，增大 kernel 的感受野。深度网络越往后面越能捕捉到物体的语义信息，这种语义信息是建立在较大的感受野基础上。已古人的例子来做解释，想必大家都知道盲人摸象这个成语的来历，每个盲人只能触摸到大象的一部分，也就是只能获得 local response，基于这些 local response，盲人们很难猜对他们到底

在摸什么。即使是一个明眼人，眼睛紧贴这大象后背看，也很难猜到看的是什么。这个例子告诉我们局部信息很难提供更高层的语义信息，因此对 feature map 降维，进而增大后面各层 kernel 的感受野是一件很重要的事情。另外一点值得注意：pooling 也可以提供一些旋转不变性。

8. 卷积层、池化层和激活函数层等操作是将原始数据映射到隐层特征空间的话，全连接层则起到将学到的“分布式特征表示”映射到样本标记空间的作用。

9. 全连接存在的问题：参数量过大，降低了训练的速度，且很容易过拟合。

10. 一般在全连接后会有激活函数来做分类，假设这个激活函数是一个多分类 softmax，那么全连接网络的作用就是将最后一层卷积得到的 feature map stretch 成向量，对这个向量做乘法，最终降低其维度，然后输入到 softmax 层中得到对应的每个类别的得分。

11. 由此就可以比较直观地说明了。这两者合二为一的过程我们可以探索到 GAP 的真正意义是:对整个网路在结构上做正则化防止过拟合。其直接剔除了全连接层中黑箱的特征，直接赋予了每个 channel 实际的内别意义。

实践证明其效果还是比较可观的，同时 GAP 可以实现任意图像大小的输入。但是值得我们注意的是，使用 gap 可能会造成收敛速度减慢。

global average pooling 与 average pooling 的差别就在 "global" 这一个字眼上。global 与 local 在字面上都是用来形容 pooling 窗口区域的。 local 是取 feature map 的一个子区域求平均值，然后滑动这个子区域； global 显然就是对整个 feature map 求平均值了。

12. attention 的本质

Attention 的本质是提取重要信息。即对重要信息赋予更高的权重，对不重要的信息赋予低的权重。

13. CNN+RNN

1) 不同点

CNN 进行空间扩展，神经元与特征卷积；RNN 进行时间扩展，神经元与多个时间输出计算；

RNN 可以用于描述时间上连续状态的输出，有记忆功能；CNN 则用于静态输出；

CNN 高级结构可以达到 100+深度；RNN 的深度有限。

2. 组合的意义

大量信息同时具有时间空间特性：视频，图文结合，真实的场景对话；

带有图像的对话，文本表达更具体；

视频相对图片描述的内容更完整。

14. 在 kfold 时，如果数据时序性强的话，不要打乱。如果时序性不强的话，可以打乱。还有 random_state = 42, 也是可以上分的在竞赛时。

15. 特征提取

同等类型的要做加减乘除，不同类型的看能不能做交互。

Null 值是可以的，有时比有值（如果长或宽是 0 的情况下）效果更好

16. 类别不平衡问题

1) 扩大数据集，然后对大类样本做欠采样

2) 对数据集做重采样（对大类样本做欠采样，对小类样本做过采样）

3) 尝试对模型进行惩罚

你可以使用相同的分类算法，但是使用一个不同的角度，比如你的分类任务是识别那些小类，那么可以对分类器的小类样本数据增加权值，降低大类样本的权值（这种方法其实是产生了新的数据分布，即产生了新的数据集，译者注），从而使得分类器将重点集中在小类样本身上。一个具体做法就是，在训练分类器时，若分类器将小类样本分错时额外增加分类器一个小类样本分错代价，这个额外的代价可以使得分类器更加“关心”小类样本。

4) 尝试不同的分类算法

1. Accuracy 对样本不均衡不准确，可以用 F1 算法

17. 交叉验证：

1) 简单交叉验证：

首先，我们随机的将样本数据分为两部分（比如： 70%的训练集，30%的测试集），然后用训练集来训练模型，在测试集上验证模型及参数。接着，我们再把样本打乱，重新选择训练集和测试集，继续训练数据和检验模型。最后我们选择损失函数评估最优的模型和参数

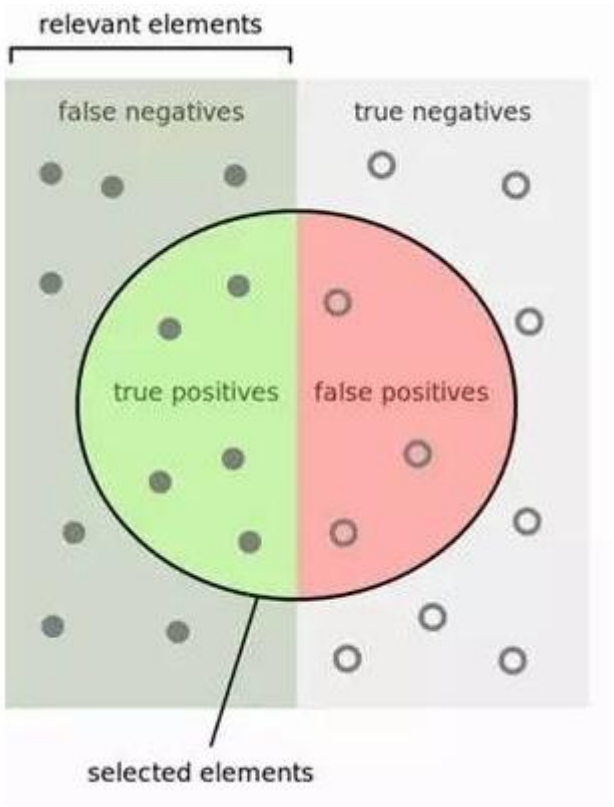
2) S 折交叉验证:

S 折交叉验证会把样本数据随机的分成 S 份，每次随机的选择 S-1 份作为训练集，剩下的 1 份做测试集。当这一轮完成后，重新随机选择 S-1 份来训练数据。若干轮（小于 S）之后，选择损失函数评估最优的模型和参数。

3) 留 1 交叉验证:

4) 此时 S 等于样本数 N，这样对于 N 个样本，每次选择 N-1 个样本来训练数据，留一个样本来验证模型预测的好坏。此方法主要用于样本量非常少的情况，比如对于普通适中问题，N 小于 50 时，我一般采用留一交叉验证。

18. 模型评估

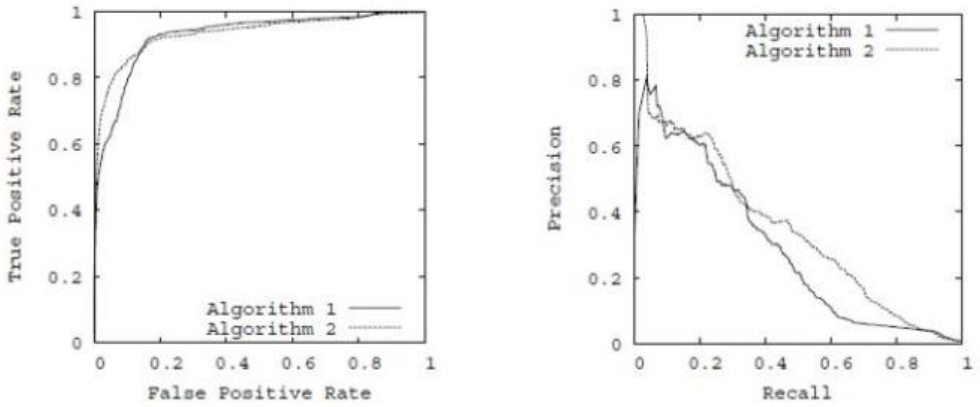


Precision: $P = \frac{TP}{TP + FP}$ recall: $R = \frac{TP}{TP + FN}$ 特异性（specificity）：$S = \frac{TN}{FP + TN}$ F1 score: $\frac{2}{F_1} = \frac{1}{P} + \frac{1}{R}$ 灵敏度(True positive rate)：

$$TPR = \frac{TP}{TP + FN}$$ 特异度（False positive rate）：$$FPR = \frac{FP}{FP + TN}$$

ROC 曲线和 PR 曲线如下：

以TPR为y轴，以FPR为x轴，我们就直接得到了RoC曲线。从FPR和TPR的定义可以理解，TPR越高，FPR越小，我们的模型和算法就越高效。也就是画出来的RoC曲线越靠近左上越好。如下图左图所示。从几何的角度讲，RoC曲线下方的面积越大越大，则模型越优。所以有时候我们用RoC曲线下的面积，即AUC（Area Under Curve）值来作为算法和模型好坏的标准。



(a) Comparison in ROC space (b) Comparison in PR space

以精确率为y轴，以召回率为x轴，我们就得到了PR曲线。仍然从精确率和召回率的定义可以理解，精确率越高，召回率越高，我们的模型和算法就越高效。也就是画出来的PR曲线越靠近右上越好。如上图右图所示。

使用RoC曲线和PR曲线，我们就能很方便的评估我们的模型的分类能力的优劣了。

19. 自然语言先用词袋(bag of words)来表示句子，然后用 google 的 word2vec，现在有用 bert 来做嵌入。Bert 可以考虑到相同词在不同位置会有不同含义等信息。
20. Embedding 是一种典型的利用无监督提升监督问题的解决效果的手段。

word2vec 训练时权重矩阵过大怎么加速训练？

1) 高频词抽样

对于 the 这样的单词：

(1) 当我们得到成对的单词训练样本时，('fox', 'the')这样的训练样本并不会给我们提供关于'fox'更多的语义信息，因为'the'在每个单词的上下文中几乎都会出现

(2) 但训练样本中有许多类似('the', ...)这样的样本，这样的样本数量远远超过了我们学习'the'这个词向量所需的训练样本。

Word2vec 通脱采样来解决这样的高频词。基本思想如下：对于我们在训练原始文本中遇到的每一个单词，它们都有一定概率被我们从文本中删除，而这个删除的概率与单词的频率相关。

如果我们设置窗口的大小为 10，并且从我们的文本中删除所有的"the"，那会有下面的结果：

1)) 由于我们删除了"the"，那么在训练集合中，"the"这个词永远也不会出现在我们的上下文窗口中。

2)) 当"the"作为 input word 时，我们的训练样本数量至少减少 10 个。

代表着保留某个单词的概率：

$$P(w_i) = (\sqrt{\frac{Z(w_i)}{0.001}} + 1) \times \frac{0.001}{Z(w_i)}$$

2) 负采样的作用：

1. 提高训练速度
2. 改善所得到的词向量的质量

负采样每次让一个训练样本仅仅更新一小部分的权重。

原文作者建议 negative words 数量：对于小规模数据集，选择 5-20 个 negative words 会比较好。对于大数据集，选择 2-5 个会比较好。

3) How to select negative words?

一个单词被选作 negative sample 的概率跟它出现的频次有关，出现频次越高的单词越容易背选作 negative words

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n}(f(w_j)^{3/4})}$$

f(w_i)代表着单词出现的频次。其中 3/4 完全是根据经验而来。

随机生成一个数（1 - 1 亿），然后在 unigram table 中根据这个 index 找到相应的 word index。

4) Negative sampling 也是采用了二元逻辑回归来求解模型参数。

Negative sampling 重点：https://www.cnblogs.com/pinard/p/7249903.html

5) Hierarchical softmax

Word2vec 最大问题在于从隐藏层到输出的 softmax 层的计算量很大。因为要计算所有词（V 是词汇表的大小）的 softmax 概率，再去找概率最大的值。

改进如下：

1. 对于从输入层到隐藏层的映射，没有采取神经网络的线性变化加激活函数，而是采用简单的对所有输入向量求和取平均值。
2. 从隐藏层到输出层，采用 huffman tree（叶子节点为词汇表大小）来代替计算 softmax。其中根节点的词向量对应我们的投影后的词向量，而所有叶子节点就类似 softmax 输出层的神经元，叶子节点个数就是词汇表的大小。
3. 用 sigmoid 来判别该向量是沿着左子树还是右子树走（左子树代表负样本，右子树代表正样本，visa verse），
6) Huffman 的好处
1) 由于是二叉树，之前计算量为 V，现在变成了 log(2)V
2) 由于高频的词靠近树根，这样我们更容易找到高频词
huffman:带权路径长度最短为目标，可以用 log(v)长度的编码表示。
在 word2vec 中，由于使用的是随机梯度上升法，因此仅仅只使用一个样本来更新梯度，这样是为了减少梯度计算量。
Skip-gram 使用期望 P(x_w/x_i)最大，i=1,2...2c 而不用 p(x_i/x_w)。因为这样整体的迭代会更加的均衡。也因为这个原因 skip-gram 模型并没有跟 cbow 一样对输入进行迭代更新，而是对 2c 个输出进行迭代更新。
Hierarchical Softmax:https://www.cnblogs.com/pinard/p/7243513.html
7) Huffman 的劣势：
当我们的训练样本里的中心词 w 是一个很生僻的词时，那就得在 huffman 树中辛苦的走下去。能不能不用搞这么复杂的一颗 huffman 树，将模型变得更加简单呢？ 所以我们可以用 negative sampling。
8) 为什么一个单词能预测它周围的单词，这个的基本假设是什么？
如果一个单词是类似的，那么他们周围的单词也是类似的。对于周围单词相似的，我们希望这单词也是相似的。
21. 线性回归
1）线性回归的模型和损失函数
2）线性回归的算法
一种是梯度下降法和另一种是最小二乘法
什么是梯度？
在微积分里面，对多元函数的参数求偏导数，把求得的各个参数的偏导数以向量的形式写出来就是梯度。
梯度的意义？
在几何中，其就是函数变化增加最快的地方。具体的说就是沿着梯度的方向，函数增加最快的地方。
梯度下降是局部最优解。但是如果损失函数是凸函数，梯度下降得到的解就一定是全局最优解。
梯度下降的一些概念：learning rate, feature, model(hypothesis function), and loss function
梯度的下降的算法调优：算法的步长选择（步长过大可能会错过最优解，步长太小收敛速度太慢，有可能陷入局部最优解），初始值选择（不同的初始值获得的最小值也有不同，因为陷入局部最优解，当然凸函数除外。因此我们需要多次不同的初始值运行算法，选择最好的一个），归一化（样本中有的特征取值范围不一样，所以迭代很慢，我们可以进行归一化（均值为 0 方差为 1））
梯度下降和最小二乘的不同：

梯度下降需要步长，最小二乘不需要。
梯度下降是迭代求解，最小二乘是计算解析解。
如果样本量不算大，且存在解析解，最小二乘计算速度很快。如果样本量很大或者没有解析解，用梯度下降方法。
梯度下降和牛顿法都是用迭代求解，但梯度下降用的是梯度方法，牛顿法用的是海森矩阵的逆矩阵或伪矩阵求解。用牛顿法收敛更快，但每次迭代的时间比梯度下降更长。
如果拟合函数不是线性的，这时无法使用最小二乘法

$$\theta = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y}$$

梯度下降法： https://www.cnblogs.com/pinard/p/5970503.html
最小二乘法：https://www.cnblogs.com/pinard/p/5976811.html

3）线性回归的 L1 正则化通常称为 Lasso 回归，它和一般线性回归的区别是在损失函数上增加了一个 L1 正则化的项，L1 正则化的项有一个常数系数 α来调节损失函数的均方差项和正则化项的权重，具体 Lasso 回归的损失函数表达式如下：

$$J(\theta) = \frac{1}{2}(\mathbf{X}\theta - \mathbf{Y})^T(\mathbf{X}\theta - \mathbf{Y}) + \alpha||\theta||_1$$

其中 $||\theta||_1$ 为L1范数。

Lasso 回归可以使得一些特征的系数变小，甚至还是一些绝对值较小的系数直接变为 0。增强模型的泛化能力

Lasso 回归的求解办法一般有坐标轴下降法（coordinate descent）和最小角回归法(Least Angle Regression)

Ridge 回归(loss function + L2 正则项)在不抛弃任何一个特征的情况下，缩小了回归系数，使得模型相对而言比较的稳定，但和 Lasso 回归比，这会使得模型的特征留的特别多，模型解释性差。Ridge 回归一般用最小二乘法。

$$\theta = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \alpha\mathbf{E})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y}$$

线性回归：https://www.cnblogs.com/pinard/p/6004041.html

22. 逻辑回归

1) 逻辑回归的模型

逻辑回归就是在线性回归后面再加一个 sigmoid 函数.

2) 逻辑回归的损失函数

$$P(y=1|x,\theta)=h_\theta(x)$$

$$P(y=0|x,\theta)=1-h_\theta(x)$$

把这两个式子写成一个式子，就是：

$$P(y|x,\theta)=h_\theta(x)^y(1-h_\theta(x))^{1-y}$$

其中y的取值只能是0或者1。

得到了y的概率分布函数表达式，我们就可以用似然函数最大化来求解我们需要的模型系数θ。

为了方便求解，这里我们用对数似然函数最大化，对数似然函数取反即为我们的损失函数 (θ)。其中：

似然函数的代数表达式为：

$$L(\theta)=\prod_{i=1}^{m}(h_\theta(x^{\,i}))^{y^{(i)}}(1-h_\theta(x^{\,i}))^{1-y^{(i)}}$$

3) 损失函数的优化方法

$$\theta=\theta-\alpha X^T(h_\theta(X)-Y)$$

4) 二元逻辑回归的正则化

对 L2 正则项：正常的梯度下降法

对 L1 正则项：可以用 coordinate descent and 最小角回归

5) 多元逻辑回归：

对于 K 分类有两种分类方法：1. K 个分类并不互斥（对每个分类用 sigmoid 做二分类）2. K 个分类互斥，那直接用 softmax 函数来做 K 分类。

$$p(c=k|x;\theta)=\frac{e^{\theta_k^T x}}{\sum_{l=1}^{K}e^{\theta_l^T x}},k=1,2,\cdots,K$$

似然函数：

$$L(\theta)=\prod_{i=1}^{m}\prod_{k=1}^{K}p(c=k|x^{(i)};\theta)^{y_k^{(i)}}$$
$$=\prod_{i=1}^{m}\prod_{k=1}^{K}\left(\frac{e^{\theta_k^T x^{(i)}}}{\sum_{l=1}^{K}e^{\theta_l^T x^{(i)}}}\right)^{y_k^{(i)}}$$

对数似然：

$$J_m(\theta)=\ln L(\theta)=\sum_{i=1}^{m}\sum_{k=1}^{K}y_k^{(i)}\left(\theta_k^T x^{(i)}-\ln\sum_{l=1}^{K}e^{\theta_l^T x^{(i)}}\right)$$

对于单个样本：

$$J(\theta)=\sum_{k=1}^{K}y_k\left(\theta_k^T x-\ln\sum_{l=1}^{K}e^{\theta_l^T x}\right)$$

随机梯度：

$$\frac{\partial J(\theta)}{\partial\theta_k}=(y_k-p(y_k|x;\theta))x$$

感知机算法原理

1) 感知机模型：

感知机的模型就是尝试找到一条直线或一个超平面，能够把所有的男孩和女孩隔离开。如果不能分开，就是线性不可分。感知机不可利用。

y = sign(theta * x) 其中 sign(t) = {-1 t <0, 1 t>0}

2) 感知机 loss function

$$J(\theta) = -\sum_{x_i \in M} y^{(i)}\theta \bullet x^{(i)}$$

3) 感知机优化：

$$\theta = \theta + \alpha y^{(i)} x^{(i)}$$

其中α为步长，$y^{(i)}$为样本输出1或者-1，$x^{(i)}$为(n+1)x1的向量。

23. 决策树算法原理

决策树算法既可以作为分类算法也可以作为回归算法，同时也适合集成学习，比如随机森林。先决定哪个特征，在决定哪个特征。怎么准确的定量选择这个标准就是决策树算法的关键了。我们用信息的熵来度量决策树的决策选择过程。

$$H(X) = -\sum_{i=1}^{n} p_i log p_i$$

$$H(X,Y) = -\sum_{i=1}^{n} p(x_i, y_i) log p(x_i, y_i)$$

$$H(X|Y) = -\sum_{i=1}^{n} p(x_i, y_i) log p(x_i|y_i) = \sum_{j=1}^{n} p(y_j) H(X|y_j)$$

信息增益：I(X,Y) = H(X)- H(X/Y)

ID3 算法就是用信息增益来度量决策树的选择过程。

ID3 算法的缺陷：

1) ID3 没有考虑连续特征

2) ID3 采用信息增益来判断。但是取值比较多的特征比取值比较少的特征信息增益大。

3) ID3 算法对缺失值的情况没有考虑

4) 没有考虑过拟合的问题

C4.5 算法解决上述缺陷：

1) 对连续值进行离散化，然后分别计算以该点作为二元分类点时的信息增益。选择最大得那个。但于离散不同的是，该属性后面还可以参与子节点的产生选择过程。

2) 我们用信息增益比来描述 I_r(D,A)

$$I_R(D,A)=\frac{I(A,D)}{H_A(D)}$$

其中D为样本特征输出的集合，A为样本特征，对于特征熵$H_A(D)$，表达式如下：

$$H_A(D)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}log_2\frac{|D_i|}{|D|}$$

其中n为特征A的类别数，D_i为特征A的第i个取值对应的样本个数。$|D|$为样本个数。

3) 解决方法有两种：一是在样本某些特征缺失的情况下选择划分的属性，二是选定了划分属性，对于在该属性上缺失特征的样本的处理。

对于第一个子问题，对于某一个有缺失特征值的特征 A。C4.5 的思路是将数据分成两部分，对每个样本设置一个权重（初始可以都为 1），然后划分数据，一部分是有特征值 A 的数据 D1，另一部分是没有特征 A 的数据 D2. 然后对于没有缺失特征 A 的数据集 D1 来和对应的 A 特征的各个特征值一起计算加权重后的信息增益比，最后乘上一个系数，这个系数是无特征 A 缺失的样本加权后所占加权总样本的比例。

对于第二个子问题，可以将缺失特征的样本同时划分入所有的子节点，不过将该样本的权重按各个子节点样本的数量比例来分配。比如缺失特征 A 的样本 a 之前权重为 1，特征 A 有 3 个特征值 A1,A2,A3。3 个特征值对应的无缺失 A 特征的样本个数为 2,3,4.则 a 同时划分入 A1，A2，A3。对应权重调节为 2/9,3/9, 4/9。

4) 引入正则化系数进行初步的剪枝。

C4.5 算法的缺陷：

1) 过拟合有优化空间。剪枝有两种思路：一是预剪枝，二是先生成决策树，再通过交叉验证来剪枝。

2) C4.5 生成的是多叉树

3) C4.5 只能用于分类，不能用于回归

4) 由于使用了熵模型，里面有大量的对数运算，如果是连续值还有大量的排序运算。

参考：https://www.cnblogs.com/pinard/p/6050306.html

CART 分类树算法的最优特征选择方法：

CART 分类树算法使用基尼系数来代替信息增益比，基尼系数代表了模型的不纯度，基尼系数越小，则不纯度越低，特征越好。这和信息增益(比)是相反的。

对于 K 分类问题：

$$Gini(p)=\sum_{k=1}^{K}p_k(1-p_k)=1-\sum_{k=1}^{K}p_k^2$$

对于 2 分类问题：

$$Gini(p)=2p(1-p)$$

对于个给定的样本D,假设有K个类别，第k个类别的数量为C_k,则样本D的基尼系数表达式为：

$$Gini(D) = 1 - \sum_{k=1}^{K}(\frac{|C_k|}{|D|})^2$$

特别的，对于样本D,如果根据特征A的某个值a,把D分成D1和D2两部分，则在特征A的条件下，D的基尼系数表达式为：

$$Gini(D,A) = \frac{|D_1|}{|D|}Gini(D_1) + \frac{|D_2|}{|D|}Gini(D_2)$$

因为基尼系数与熵之半的曲线很接近，所以基尼系数可以作为熵的近似代替。而且 CART 分类树算法每次仅仅对某个特征值进行二分，而不是多分。

CART 分类树算法对于连续特征和离散特征处理的改进

对于连续特征处理与 C4.5 不同的是该属性后面还可以参与子节点的产生选择过程。
对于离散特征处理是不停的二分离散特征。
1) CART 回归树和 CART 分类树的建立和预测的区别主要有下面两点：
连续值的处理方法不同和决策树建立后做预测的方式不同
2) 对于连续值的处理，CART 分类树采用的是用基尼系数的大小来度量特征的各个划分点的优劣情况。对于回归模型我们用的是和方差的度量方式。

$$\underbrace{min}_{A,s}\left[\underbrace{min}_{c_1}\sum_{x_i \in D_1(A,s)}(y_i - c_1)^2 + \underbrace{min}_{c_2}\sum_{x_i \in D_2(A,s)}(y_i - c_2)^2\right]$$

其中，c_1为D1数据集的样本输出均值，c_2为D2数据集的样本输出均值。

3) 对于决策树建立后做预测的方式，上面讲到了 CART 分类树采用叶子节点里概率最大的类别作为当前节点的预测类别。而回归树输出不是类别，它采用的是用最终叶子的均值或者中位数来预测输出结果。

CART 树算法的剪枝

CART 采用的办法是后剪枝法，即先生成决策树，然后产生所有可能的剪枝后的 CART 树，然后使用交叉验证来检验各种剪枝的效果，选择泛化能力最好的剪枝策略。
剪枝的损失函数度量(在剪枝过程中，对于任意的一刻)：
重点：https://www.cnblogs.com/pinard/p/6053344.html

集成学习

集成学习可以用于分类问题集成，回归问题集成，特征选取集成，异常点检测集成等等
集成学习有两个主要的问题需要解决，一是如何得到若干个个体学习器，第二是如何选择一种结合策略

个体学习器

个体学习器都是同质的或者是异质的。
常用的同质学习器有 cart 决策树和神经网络。同质学习器根据依赖关系可以分为强依赖关系（个体学习器基本都需要串行生成，代表算法是 boosting 系列算法）和不存在强依赖关系

（个体学习器可以并行生成，代表算法是 bagging 和随机森林（random forest））。

Boosting

boosting 算法的工作机制是首先从训练集用初始权重训练出一个弱学习器 1，根据弱学习器的学习误差率来更新训练样本的权重，使得之前弱学习器 1 学习误差率高的训练样本点的权重变高，使得这些误差率高的点在后面的样本弱学习器 2 中得到更多的重视。
Boosting 著名算法有 adaboost 算法和提升树(boosting tree). 提升树算法里面应用最广泛的是梯度提升树(Gradient Boosting Tree)

Bagging

bagging 的个体弱学习器的训练集是通过随机采样得到的.通过 T 次的随机采样，我们就可以得到 T 个采样集，对于这 T 个采样集，我们可以分别独立的训练出 T 个弱学习器.
随机森林是 bagging 的特异进阶版。其的弱学习器都是 cart 决策树，而且其在 bagging 的随机采样基础上，又加了特征的随机选择。
结合策略
1. 平均法
对于回归问题经常用平均法：
1) $$H(x)=\frac{1}{T}\sum_{1}^{T}h_i(x)$$ 2) $$H(x)=\sum_{i=1}^{T}w_ih_i(x)$$
2. 投票法
对于分类问题经常用投票法。
1) 相对多数投票法 2) 绝对多数投票法，如果没有超过一半（自己规定）就拒绝预测。 3) 加权投票法
3. 学习法
上两节的方法都是对弱学习器的结果做平均或者投票，相对比较简单，但是可能学习误差较大，于是就有了学习法这种方法，对于学习法，代表方法是 stacking，当使用 stacking 的结合策略时， 我们不是对弱学习器的结果做简单的逻辑处理，而是再加上一层学习器，也就是说，我们将训练集弱学习器的学习结果作为输入，将训练集的输出作为输出，重新训练一个学习器来得到最终结果。

在这种情况下，我们将弱学习器称为初级学习器，将用于结合的学习器称为次级学习器。对于测试集，我们首先用初级学习器预测一次，得到次级学习器的输入样本，再用次级学习器预测一次，得到最终的预测结果。

集成学习之 Adaboost

Adaboost 算法原理

boosting 算法的工作机制是首先从训练集用初始权重训练出一个弱学习器 1，根据弱学习器的学习误差率来更新训练样本的权重，使得之前弱学习器 1 学习误差率高的训练样本点的权重变高，使得这些误差率高的点在后面的样本弱学习器 2 中得到更多的重视。
Adaboost 可以用作分类也可以用作回归。

不过有几个具体的问题 Boosting 算法没有详细说明。

1）如何计算学习误差率 e?

2) 如何得到弱学习器权重系数 α?

3）如何更新样本权重 D?

4) 使用何种结合策略？

Adaboost 损失函数和分类器算法

Adaboost 是加法模型，如下：

$$f_{k-1}(x)=\sum_{i=1}^{k-1}\alpha_i G_i(x)$$

$$f_k(x)=\sum_{i=1}^{k}\alpha_i G_i(x)$$

$$f_k(x)=f_{k-1}(x)+\alpha_k G_k(x)$$

损失函数是指数函数，如下：

$$\underbrace{arg\ min}_{\alpha,G}\sum_{i=1}^{m}exp(-y_i f_k(x))$$

利用前向分布学习算法可以得到损失函数为：

$$(\alpha_k,G_k(x))=\underbrace{arg\ min}_{\alpha,G}\sum_{i=1}^{m}exp[(-y_i)(f_{k-1}(x)+\alpha G(x))]$$

然后推导到第 k 个弱学习器为：

$$G_k(x)=\underbrace{arg\ min}_{G}\sum_{i=1}^{m}w_{ki}^{'}I(y_i\neq G(x_i))$$

分类误差率为：

$$e_k = \frac{\sum_{i=1}^{m} w'_{ki} I(y_i \neq G(x_i))}{\sum_{i=1}^{m} w'_{ki}} = \sum_{i=1}^{m} w_{ki} I(y_i \neq G(x_i))$$

分类器系数权重：

$$\alpha_k = \frac{1}{2} log \frac{1 - e_k}{e_k}$$

更新样本权重系数：

$$w_{k+1,i} = \frac{w_{ki}}{Z_K} exp(-\alpha_k y_i G_k(x_i)) \quad i = 1, 2, \ldots m$$

这里Z_k是规范化因子

$$Z_k = \sum_{i=1}^{m} w_{ki} exp(-\alpha_k y_i G_k(x_i))$$

最终构建分类器为

$$f(x) = sign(\sum_{k=1}^{K} \alpha_k G_k(x))$$

对于多分类，其和二分类很相似，唯一不同就是

$$\alpha_k = \frac{1}{2} log \frac{1 - e_k}{e_k} + log(R - 1)$$

Adaboost 回归问题的算法流程：

输入为样本集$T = \{(x,y_1),(x_2,y_2),...(x_m,y_m)\}$，，弱学习器算法，弱学习器迭代次数K。

输出为最终的强学习器$f(x)$

1) 初始化样本集权重为

$$D(1) = (w_{11}, w_{12}, ...w_{1m});\ \ w_{1i} = \frac{1}{m};\ \ i = 1,2...m$$

2) 对于k=1,2，...K:

a) 使用具有权重D_k的样本集来训练数据，得到弱学习器$G_k(x)$

b) 计算训练集上的最大误差

$$E_k = max|y_i - G_k(x_i)|\ i = 1,2...m$$

c) 计算每个样本的相对误差:

如果是线性误差，则$e_{ki} = \frac{|y_i - G_k(x_i)|}{E_k}$；

如果是平方误差，则$e_{ki} = \frac{(y_i - G_k(x_i))^2}{E_k^2}$

如果是指数误差，则$e_{ki} = 1 - exp(\frac{-|y_i - G_k(x_i)|}{E_k})$

d) 计算回归误差率

$$e_k = \sum_{i=1}^{m} w_{ki} e_{ki}$$

c) 计算弱学习器的系数

$$\alpha_k = \frac{e_k}{1 - e_k}$$

d) 更新样本集的权重分布为

$$w_{k+1,i} = \frac{w_{ki}}{Z_k} \alpha_k^{1-e_{ki}}$$

这里Z_k是规范化因子

$$Z_k = \sum_{i=1}^{m} w_{ki} \alpha_k^{1-e_{ki}}$$

3) 构建最终强学习器为：

$$f(x) = G_{k^*}(x)$$

其中，$G_{k^*}(x)$是所有$ln\frac{1}{\alpha_k}, k = 1,2,....K$的中位数值对应序号$k^*$对应的弱学习器。

Adaboost 算法的正则化

为了防止过拟合，我们会加入正则项，其被称为 learning rate，定义为 v，之前的弱学习器迭代为

$$f_k(x) = f_{k-1}(x) + \alpha_k G_k(x)$$

现在为

$$f_k(x) = f_{k-1}(x) + \nu\alpha_k G_k(x)$$

其中 0 < v < 1

Adaboost 的总结

优点：

1) Adaboost 作为分类器时，分类精度很高

2) Adaboost 不容易发生过拟合

3) 可以使用各种回归分类模型来构建学习器，非常灵活

缺点：

对异常数据非常敏感，异常样本在迭代中可能会获得较高的的权重，影响最终的强学习器的预测准确性。

http://www.6aiq.com/article/1535805751770?p=1&m=0

https://www.cnblogs.com/pinard/p/6133937.html

集成学习之 Bagging

Bagging 算法原理

Bagging 算法和 GBDT 的采样是不同的。Bagging 算法的子采样是放回采样。GBDT 是无放回采样。

对于一个样本，它在某一次含m个样本的训练集的随机采样中，每次被采集到的概率是$\frac{1}{m}$。不被采集到的概率为$1-\frac{1}{m}$。如果m次采样都没有被采集中的概率是$(1-\frac{1}{m})^m$。当$m \to \infty$时，$(1-\frac{1}{m})^m \to \frac{1}{e} \simeq 0.368$。也就是说，在bagging的每轮随机采样中，训练集中大约有36.8%的数据没有被采样集采集中。

这些数据被称为袋外数据（out of Bag），这些数据没有参于模型的拟合，因为被用来检测模型的泛化能力。

Bagging 算法有很强的泛化能力，对于降低模型的方差很有作用。当然模型的拟合程度会差一点，偏移会大一点。

随机森林算法：

随机森林跟 bagging 算法的区别是：

1) Random forest 用 cart 决策树来作为弱学习器

2) 在建立 car 树时，不是从所有特征样本（e.g. n 个特征）中选择最优的特征作为节点，而是先随机从样本中抽取一部分特征（e.g. n_sub 个特征），然后从其选取最优的那个特征。这样做的好处是进一步增强了模型的泛化能力。

如果$n_{sub} = n$，则此时RF的CART决策树和普通的CART决策树没有区别。n_{sub}越小，则模型约健壮，当然此时对于训练集的拟合程度会变差。也就是说n_{sub}越小，模型的方差会减小，但是偏倚会增大。在实际案例中，一般会通过交叉验证调参获取一个合适的n_{sub}的值。

随机森林不光可以应用分类回归，还可以应用于特征转换，异常点检测，特征提取。

Isolation Forest (主要应用于异常点检测)

其和 RF 不同处：

1) 采样时只需随机采样一小部分
2) 建造 cart 决策树时随机选取特征，根据特征随机选取特征值。
3) 建造树的深度不需要太高

对于异常点的判断，则是将测试样本点x拟合到T颗决策树。计算在每颗决策树上该样本的叶子节点的深度$h_t(x)$。从而可以计算出平均高度h(x)。此时我们用下面的公式计算样本点x的异常概率:

$$s(x,m) = 2^{-\frac{h(x)}{c(m)}}$$

其中，m为样本个数。$c(m)$的表达式为：

$$c(m) = 2\ln(m-1) + \xi - 2\frac{m-1}{m},\ \xi\text{为欧拉常数}$$

s(x,m)的取值范围是[0,1],取值越接近于1，则是异常点的概率也越大。

RF 小结：

优点：

1. 训练可以高度并行化，对大数据时代，尤其重要。
2. 由于可以随机选择决策树节点划分特征，这样在高维特征时，仍然能高效的训练模型。
3. 可以得到特征们的重要程度
4. 由于采用了随机采样，训练出的模型的方差小，泛化能力强
5. 对部分特征缺失不敏感

缺点：

1. 取值划分比较多的特征容易对 rf 的决策树产生更大的影响，从而影响拟合的模型效果
2. 在某些噪音比较大的样本集上，RF 容易陷入过拟合

梯度提升树(GBDT)原理小结

GBDT 概述

GBDT 和 Adaboost 的差别是弱学习器只能用 CART 回归树模型，同时迭代思路和 Adaboost 也有所不同。

1. Adaboost 算法是通过给已有模型预测错误的样本更高的权重，使得先前的学习器做错的训练样本在后续受到更多的关注来弥补已有模型的不足

GBDT 在迭代的每一步构建一个能够沿着梯度最陡的方向降低损失的学习器来弥补已有模型的不足。

2. 经典的 Adaboost 算法只能处理采用指数损失函数的二分类问题。而 GBDT 可通过设置不同的可微损失函数来处理各类学习任务（多分类，回归，ranking）
3. Adaboost 算法对异常点（outlier）比较敏感，而 GBDT 通过引入 bagging 思想，加入正则化等方法能够有效的抵御训练数据中的噪音，有更好的健壮性。

GBDT 为什么选决策树作为基学习器的梯度提升方法

1. 决策树可以被认为是 if-then 规则的集合，易于理解，预测速度快，可解释性强
2. 决策树算法相比其他算法需要更少的特征工程，e.g 不用做特征标准化，可以很好的处理字段缺失的数据，不用关心特征间是否相互依赖等。
3. 决策树能够自动组合多个特征，它可以毫无压力的处理特征间的交互关系并且是非参数化的，因此你不不必担心异常值或者数据是否线性可分。

单独使用决策树算法时，有容易过拟合缺点。怎么解决呢？

- 通过各种方法，抑制决策树的复杂性，**降低单棵决策树的拟合能力**
- 通过梯度提升的方法集成多个决策树，则预测效果上来的同时，也能够很好的解决过拟合的问题。
  （这一点具有 bagging 的思想，降低单个学习器的拟合能力，提高方法的泛化能力。）

抑制单颗决策树的复杂度的方法有很多：

- 限制树的最大深度、限制叶子节点的最少样本数量、限制节点分裂时的最少样本数量
- 吸收 bagging 的思想对训练样本采样（subsample），在学习单颗决策树时只使用一部分训练样本
- 借鉴随机森林的思路在学习单颗决策树时只采样一部分特征
- **在目标函数中添加正则项惩罚复杂的树结构**等。

在GBDT的迭代中，假设我们前一轮迭代得到的强学习器是$f_{t-1}(x)$，损失函数是$L(y, f_{t-1}(x))$，我们本轮迭代的目标是找到一个CART回归树模型的弱学习器$h_t(x)$，让本轮的损失函数$L(y, f_t(x) = L(y, f_{t-1}(x) + h_t(x))$最小。也就是说，本轮迭代找到决策树，要让样本的损失尽量变得更小。

GBDT 和 XGboost 的区别是最优化的方法不同，GBDT 在函数空间中利用梯度下降法进行优化，而 XGboost 在函数空间中用牛顿法进行优化。同时 XGboost 有一些过拟合策略。

GBDT 的负梯度拟合

第 t 轮的第 i 个样本的损失函数的负梯度表示为：

$$r_{ti} = -\left[\frac{\partial L(y_i, f(x_i)))}{\partial f(x_i)}\right]_{f(x)=f_{t-1}\quad(x)}$$

针对每一个叶子节点，我们求出损失函数最小，也就是拟合叶子节点最好的输出值 c_tj 如下

$$c_{tj} = \underbrace{arg\,min}_{c} \sum_{x_i \in R_{tj}} L(y_i, f_{t-1}(x_i) + c)$$

这样我们就得到了本轮决策树如下：

$$h_t(x) = \sum_{j=1}^{J} c_{tj} I(x \in R_{tj})$$

最终强学习器为：

$$f_t(x) = f_{t-1}(x) + \sum_{j=1}^{J} c_{tj} I(x \in R_{tj})$$

通过损失函数的负梯度来拟合，我们找到了一种通用的拟合损失函数的方法，这样无论是回归还是分类问题，我们可以用负梯度来拟合。

GBDT 的回归算法

初始化弱学习器：我们可以用样本的 label 的平均值来做

GBDT 的分类算法

由于样本输出不是连续的值而是离散的类别，导致我们无法直接从输出类别去拟合类别输出的误差。为了解决这个我们可以用指数损失函数，此时 GBDT 退化为 Adaboost 算法。另一种方法是用类似于逻辑回归的对数似然损失函数的方法。而对于对数似然损失函数，我们又有二元分类和多元分类的区别。

对于二元GBDT，如果用类似于逻辑回归的对数似然损失函数，则损失函数为：

$$L(y, f(x)) = log(1 + exp(-yf(x)))$$

其中$y \in \{-1, +1\}$。则此时的负梯度误差为

$$r_{ti} = -\left[\frac{\partial L(y, f(x_i)))}{\partial f(x_i)}\right]_{f(x)=f_{t-1}\ \ (x)} = y_i / (1 + exp(y_i f(x_i)))$$

对于生成的决策树，我们各个叶子节点的最佳负梯度拟合值为

$$c_{tj} = \underbrace{arg\ min}_{c} \sum_{x_i \in R_{tj}} log(1 + exp(-y_i(f_{t-1}(x_i) + c)))$$

由于上式比较难优化，我们一般使用近似值代替

$$c_{tj} = \sum_{x_i \in R_{tj}} r_{ti} \Big/ \sum_{x_i \in R_{tj}} |r_{ti}|(1 - |r_{ti}|)$$

除了负梯度计算和叶子节点的最佳负梯度拟合的线性搜索，二元GBDT分类和GBDT回归算法过程相同。

对于多元分类看这个连接 https://www.cnblogs.com/pinard/p/6140514.html

GBDT 常用损失函数：

a) 如果是指数损失函数，则损失函数表达式为

$$L(y, f(x)) = exp(-yf(x))$$

其负梯度计算和叶子节点的最佳负梯度拟合参见Adaboost原理篇。

b) 如果是对数损失函数，分为二元分类和多元分类两种，参见4.1节和4.2节。

对于回归算法，常用损失函数有如下4种:

a)均方差，这个是最常见的回归损失函数了

$$L(y, f(x)) = (y - f(x))^2$$

b)绝对损失，这个损失函数也很常见

$$L(y, f(x)) = |y - f(x)|$$

对应负梯度误差为：

$$sign(y_i - f(x_i))$$

GBDT 的正则化：

第一种是和Adaboost类似的正则化项，即步长(learning rate)。定义为ν,对于前面的弱学习器的迭代

$$f_k(x) = f_{k-1}(x) + h_k(x)$$

如果我们加上了正则化项，则有

$$f_k(x) = f_{k-1}(x) + \nu h_k(x)$$

GBDT 小结：

GBDT 主要的优点有：

1) 可以灵活处理各种类型的数据，包括连续值和离散值。

2) 在相对少的调参时间情况下，预测的准确率也可以比较高。这个是相对 SVM 来说的。

3）使用一些健壮的损失函数，对异常值的鲁棒性非常强。比如 Huber 损失函数和 Quantile 损失函数。

GBDT 的主要缺点有：

1)由于弱学习器之间存在依赖关系，难以并行训练数据。不过可以通过自采样的 SGBT 来达到部分并行。

原文：https://www.cnblogs.com/pinard/p/6140514.html

XGBoost 算法原理小结

XGboost 是 GBDT 的一种高效实现。主要做了三方面优化：

1. 算法本身的优化：在算法的弱学习器选择上，能选择其他模型。对损失函数加了正则化部分。对损失函数的误差部分做了泰勒二阶展开，而 GBDT 只做了负梯度，即泰勒一阶。

2. 算法的运行效率优化。可以并行计算。
3. 健壮性的优化。对于缺失值的特征，通过枚举所有缺失值在当前节点是进入左子树还是右子树来决定缺失值的处理方式

这里我们总结下XGBoost的算法主流程，基于决策树弱分类器。不涉及运行效率的优化和健壮性优化的内容。

输入是训练集样本$I=\{(x,y_1),(x_2,y_2),...(x_m,y_m)\}$，最大迭代次数T, 损失函数L， 正则化系数$\lambda,\gamma$。

输出是强学习器f(x)

对迭代轮数t=1,2,...T有：

1) 计算第i个样本(i-1,2,..m)在当前轮损失函数L基于$f_{t-1}(x_i)$的一阶导数g_{ti}，二阶导数h_{ti},计算所有样本的一阶导数和$G_t=\sum_{i=1}^{m}g_{ti}$,二阶导数和$H_t=\sum_{i=1}^{m}h_{ti}$

2) 基于当前节点尝试分裂决策树，默认分数score=0

对特征序号 k=1,2...K:

a) $G_L=0,H_L=0$

b.1) 将样本按特征k从小到大排列，依次取出第i个样本，依次计算当前样本放入左子树后，左右子树一阶和二阶导数和：

$$G_L=G_L+g_{ti},G_R=G-G_L$$

$$H_L=H_L+h_{ti},H_R=H-H_L$$

b.2) 尝试更新最大的分数：

$$score=max(score,\frac{1}{2}\frac{G_L^2}{H_L+\lambda}+\frac{1}{2}\frac{G_R^2}{H_R+\lambda}-\frac{1}{2}\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}-\gamma)$$

3) 基于最大score对应的划分特征和特征值分裂子树。

4) 如果最大score为0，则当前决策树建立完毕，计算所有叶子区域的w_{tj}，得到弱学习器$h_t(x)$，更新强学习器$f_t(x)$,进入下一轮弱学习器迭代.如果最大score不是0，则转到第2)步继续尝试分裂决策树。

参考文章：https://www.cnblogs.com/pinard/p/10979808.html

EM(Expectation Maximum)算法：

其是一个基础算法，是隐式马尔科夫算法（HMM）， LDA 主题模型的基础。
我们经常从样本观察数据中，找出样本模型的参数。最长用的方法就是极大化模型分布的对数似然函数。但是一些观察数据有未观察到的隐含数据，此时我们有隐含数据和模型参数未知，所以用 EM 算法。
参考文章：https://www.cnblogs.com/pinard/p/6912636.html

KNN（k-nearest neighbors）原理小结：

KNN 可以做分类和回归。
KNN 做回归和分类的主要区别在于最后做预测时候的决策方式不同。KNN 做分类预测时，一般是选择多数表决法，即训练集里和预测的样本特征最近的 K 个样本，预测为里面有最多类别数的类别。而 KNN 做回归时，一般是选择平均法，即最近的 K 个样本的样本输出的平均值作为回归预测值。由于两者区别不大，虽然本文主要是讲解 KNN 的分类方法，但思想对 KNN 的回归方法也适用。由于 scikit-learn 里只使用了蛮力实现(brute-force)，KD 树实现(KDTree)和球树

(BallTree)实现。

KNN 算法三要素：

KNN 算法我们主要要考虑三个重要的要素：k 值的选取，距离度量的方式和分类决策规则。
对于分类决策规则，一般都是使用前面提到的多数表决法。
对于 k 值的选择，没有一个固定的经验，一般根据样本的分布，选择一个较小的值，可以通过交叉验证选择一个合适的 k 值。

选择较小的 k 值，就相当于用较小的领域中的训练实例进行预测，训练误差会减小，只有与输入实例较近或相似的训练实例才会对预测结果起作用，与此同时带来的问题是泛化误差会增大，换句话说，K 值的减小就意味着整体模型变得复杂，容易发生过拟合；

选择较大的 k 值，就相当于用较大领域中的训练实例进行预测，其优点是可以减少泛化误差，但缺点是训练误差会增大。这时候，与输入实例较远（不相似的）训练实例也会对预测器作用，使预测发生错误，且 K 值的增大就意味着整体的模型变得简单。

一个极端是 k 等于样本数 m，则完全没有分类，此时无论输入实例是什么，都只是简单的预测它属于在训练实例中最多的类，模型过于简单。
距离的度量：欧式距离（二次平方差），曼哈顿距离（一次平方差），闵可夫斯基距离(Minkowski Distance)（p 次平方差）。

KNN 算法蛮力实现:

比较预测样本和所有的训练样本，然后找到 k 个最邻近的样本在做投票。特征多和样本多时，很花费时间

KNN 算法之 KD 树实现原理:

KD 树算法先建立树，然后再对测试集做预测。KD 树是 k 个特征维度的树。
其算法包括三个步骤：一是建树，二是搜索最邻近，三是预测
建树：
从 m 个样本的 n 维特征中，分别计算 n 个特征的取值的方差，用方差最大的第 k 维特征 n_k 来作为根节点。对于这个特征，我们选择特征 n_k 的取值的中位数 n_k_v 对应的样本作为划分点。

KNN 算法之球树实现原理:

但是当数据集是不均匀分布时，某些时候效率并不高，因为它有角。

KNN 算法小结：

KNN 的主要优点有：

1） 理论成熟，思想简单，既可以用来做分类也可以用来做回归

2） 可用于非线性分类

3） 训练时间复杂度比支持向量机之类的算法低，仅为 O(n)

4） 和朴素贝叶斯之类的算法比，对数据没有假设，准确度高，对异常点不敏感

5） 由于 KNN 方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，KNN 方法较其他方法更为适合

6）该算法比较适用于样本容量比较大的类域的自动分类，而那些样本容量较小的类域采用这种算法比较容易产生误分

KNN 的主要缺点有：

1）计算量大，尤其是特征数非常多的时候

2）样本不平衡的时候，对稀有类别的预测准确率低

3）KD 树，球树之类的模型建立需要大量的内存

4）使用懒散学习方法，基本上不学习，导致预测时速度比起逻辑回归之类的算法慢

5）相比决策树模型，KNN 模型可解释性不强

朴素贝叶斯算法原理小结

在所有的机器学习分类算法中，朴素贝叶斯和其他绝大多数的分类算法都不同。对于大多数的分类算法，比如决策树,KNN,逻辑回归，支持向量机等，他们都是判别方法，也就是直接学习出特征输出Y和特征X之间的关系，要么是决策函数$Y = f(X)$,要么是条件分布$P(Y|X)$。但是朴素贝叶斯却是生成方法，也就是直接找出特征输出Y和特征X的联合分布$P(X,Y)$,然后用$P(Y|X) = P(X,Y)/P(X)$得出。

朴素贝叶斯的统计学知识: